**一、定义和功能**

操作系统是管理和控制计算机系统硬件与软件资源，合理地使用计算机工作流程及方便用户使用的程序和数据的集合，目的是提高计算机的系统效率，增强系统的处理能力，充分发挥系统资源的利用率，方便用户使用。

操作系统主要有五大功能：处理机管理（CPU）、存储管理、设备管理，文件管理和用户接口。

操作系统的特征：并发性，共享性，虚拟性，不确定性

**二、调度（进程和线程）**

进程是用户提交给操作系统运行的最小单元。进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动，是系统进行资源调度和分配的一个独立单位。除了进程，操作系统还提供了更小粒度的调度对象-线程。线程是进程的一个实体，是CPU调度和分派的基本单位，它是比进程更小的能独立运行的基本单位。一个进程可以有多个线程，多个线程也可以并发执行。

进程的内存布局：代码块、数据区、堆、栈段。

**进程和线程的区别**：资源分配给进程，同一进程的所有线程共享该进程的所有资源。 同一进程中的多个线程共享代码段(代码和常量)，数据段(全局变量和静态变量)，扩展段(堆存储)。但是每个线程拥有自己的栈段（Java虚拟机栈），用来存放所有局部变量和临时变量、程序计数器（字节码行号指示器）、线程id（标识此线程）、寄存器组的值（原有的线程的寄存器集合的状态保存，以便将来该线程被重新切换到时能得以恢复）。

**“线程”和“进程”有什么区别和优劣呢？**

* 进程是资源分配的最小单位，线程是程序执行的最小单位。
* 进程有自己的独立地址空间，每启动一个进程，系统就会为它分配地址空间，建立数据表来维护代码段、堆栈段和数据段，这种操作非常昂贵；而线程没有独立的地址空间，它使用相同的地址空间共享数据。
* CPU切换一个线程的花费远比进程要小很多，同时创建一个线程的开销也比进程要小很多。
* 线程之间的通信更方便，同一进程下的线程共享全局变量、静态变量等数据，而进程之间的通信需要以通信的方式(IPC)进行。
* 多进程程序更健壮，生命力更强，一个进程死掉不会对另一个进程造成影响，因为他们有独立的地址空间；而多线程程序更不易维护，因为共享地址空间一个线程死掉，整个进程就死掉了；
* 多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。
* 进程对资源保护要求高，开销大，效率相对较低，线程资源保护要求不高，但开销小，效率高，可频繁切换。

协程：协程和线程一样共享堆，不共享栈，协程是一种用户态的轻量级线程，协程的调度完全由用户控制。协程拥有自己的寄存器上下文和栈。协程调度切换时，将寄存器上下文和栈保存到其他地方，在切回来的时候，恢复先前保存的寄存器上下文和栈，直接操作栈（在一个线程内执行）则基本没有内核切换的开销，可以不加锁的访问全局变量，所以上下文的切换非常快。

**协程和线程的区别**是：协程避免了无意义的调度，由此可以提高性能，但也因此，程序员必须自己承担调度的责任，同时，协程也失去了标准线程使用多CPU的能力。

一般情况下，系统按照以下流程创建一个进程：

1. 分配、初始化 PCB
2. 初始化机器寄存器
3. 拷贝、初始化内存页表
4. 从硬盘加载程序代码到内存
5. 将进程加入就绪队列
6. 进程调度时，选择该进程并切换到用户态开始执行进程

系统通过快速切换进程，让每一个进程都有一定的时间片来响应用户提交的请求；在用户的视角，好像每个进程都在同时执行一样。系统切换进程的方法叫做进程调度算法，基本的调度算法有：

（1）先来先服务（FCFS）

算法简单，但效率低；对长作业比较有利，但对短作业不利（相对SJF和高响应比）；有利于CPU繁忙型作业，而不利于I/O繁忙型作业。

（2）时间片轮转

时间片轮转调度算法主要适用于分时系统。在这种算法中，系统将所有就绪进程按到达时间的先后次序排成一个队列，进程调度程序总是选择就绪队列中第一个进程执行，即先来先服务的原则，但仅能运行一个时间片，如100ms。在使用完一个时间片后，即使进程并未完成其运行，它也必须释放出（被剥夺）处理机给下一个就绪的进程，而被剥夺的进程返回到就绪队列的末尾重新排队，等候再次运行。

在时间片轮转调度算法中，时间片的大小对系统性能的影响很大。如果时间片足够大，以至于所有进程都能在一个时间片内执行完毕，则时间片轮转调度算法就退化为先来先服务调度算法。如果时间片很小，那么处理机将在进程间过于频繁切换，使处理机的开销增大，而真正用于运行用户进程的时间将减少。因此时间片的大小应选择适当。

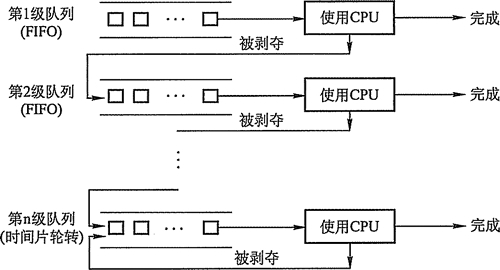
（3）短作业优先（SJF）平均等待时间、平均周转时间最少

* 该算法对长作业不利。更严重的是，如果有一长作业进入系统的后备队列，由于调度程序总是优先调度那些 (即使是后进来的）短作业，将导致长作业长期不被调度（“饥饿”现象，注意区分“死锁”。后者是系统环形等待，前者是调度策略问题）。
* 该算法完全未考虑作业的紧迫程度，因而不能保证紧迫性作业会被及时处理。
* 由于作业的长短只是根据用户所提供的估计执行时间而定的，而用户又可能会有意或无意地缩短其作业的估计运行时间，致使该算法不一定能真正做到短作业优先调度。

（4）优先级调度

* 非剥夺式优先级调度算法。当某一个进程正在处理机上运行时，即使有某个更为重要或紧迫的进程进入就绪队列，仍然让正在运行的进程继续运行，直到由于其自身的原因而主动让出处理机时（任务完成或等待事件），才把处理机分配给更为重要或紧迫的进程。
* 剥夺式优先级调度算法。当一个进程正在处理机上运行时，若有某个更为重要或紧迫的进程进入就绪队列，则立即暂停正在运行的进程，将处理机分配给更重要或紧迫的进程。

（5）多队列轮转法（优先级+时间片轮转）



算法的实现思想：

1. 应设置多个就绪队列，并为各个队列赋予不同的优先级，第1级队列的优先级最高，第2级队列次之，其余队列的优先级逐次降低。
2. 赋予各个队列中进程执行时间片的大小也各不相同，在优先级越高的队列中，每个进程的运行时间片就越小。例如，第2级队列的时间片要比第1级队列的时间片长一倍， ……第i+1级队列的时间片要比第i级队列的时间片长一倍。
3. 当一个新进程进入内存后，首先将它放入第1级队列的末尾，按FCFS原则排队等待调度。当轮到该进程执行时，如它能在该时间片内完成，便可准备撤离系统；如果它在一个时间片结束时尚未完成，调度程序便将该进程转入第2级队列的末尾，再同样地按FCFS 原则等待调度执行；如果它在第2级队列中运行一个时间片后仍未完成，再以同样的方法放入第3级队列……如此下去，当一个长进程从第1级队列依次降到第 n 级队列后，在第 n 级队列中便釆用时间片轮转的方式运行。
4. 仅当第1级队列为空时，调度程序才调度第2级队列中的进程运行；仅当第1 ~ (i-1)级队列均为空时，才会调度第i级队列中的进程运行。如果处理机正在执行第i级队列中的某进程时，又有新进程进入优先级较高的队列（第 1 ~ (i-1)中的任何一个队列），则此时新进程将抢占正在运行进程的处理机，即由调度程序把正在运行的进程放回到第i级队列的末尾，把处理机分配给新到的更高优先级的进程。

多级反馈队列的优势有：

* + 终端型作业用户：短作业优先。
  + 短批处理作业用户：周转时间较短。
  + 长批处理作业用户：经过前面几个队列得到部分执行，不会长期得不到处理。

三、通信

线程之间共享内存，但拥有各自不同的运行栈；进程之间则相互隔离。线程之间并发需要解决的是线程同步问题，进程之间则是通信问题。

临界区：我们把互斥执行的的程序段成为临界区（临界资源是一次仅允许一个进程使用的共享资源）。

四、死锁

* **什么是死锁？**

死锁是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。

例如：如果线程A锁住了记录1并等待记录2，而线程B锁住了记录2并等待记录1，这样两个线程就发生了死锁现象。

* **产生死锁的条件**

虽然进程在运行过程中，死锁的发生也必须具备一定的条件，死锁的发生必须具备以下四个**必要条件**：

**(1)互斥条件：**指进程对所分配到的资源进行排它性使用，也就是说在一段时间内某资源只由一个进程占用。如果此时还有其它进程请求资源，则请求者只能等待，直至占有资源的进程用完后被释放。

(2**)请求和保持条件：**指进程已经保持至少一个资源，但又提出了新的资源请求，而该资源已被其它进程占有，此时请求进程阻塞，但又对自己已获得的其它资源保持不放。

**(3)不剥夺条件：**指进程已获得的资源，在未使用完之前，不能被剥夺，只能在使用完时由自己释放。

**(4)环路等待条件：**指在发生死锁时，必然存在一个进程——资源的环形链，即进程集合{P0，P1，P2，···，Pn}中的P0正在等待一个P1占用的资源；P1正在等待P2占用的资源，……，Pn正在等待已被P0占用的资源。类似于死循环！

* **产生原因**

系统中的资源可以分为两类：

1.可剥夺资源，是指某进程在获得这类资源后，该资源可以再被其他进程或系统剥夺，如优先权高的进程可以剥夺优先权低的进程的处理机，内存也是一样，CPU和主存均属于可剥夺性资源；

2.不可剥夺资源，当系统把这类资源分配给某进程后，再不能强行收回，只能在进程用完后自行释放，如磁带机、打印机等。

死锁的**产生原因**有如下**三个：(1)竞争不可剥夺资源引起进程死锁**

在系统中所配置的不可剥夺资源，由于它们的数量不能满足这些进程运行的需要，会使进程在运行过程中，因争夺这些资源而陷于僵局。

**(2)竞争临时资源**，临时资源包括硬件中断、信号、消息、缓冲区内的消息等，通常消息通信顺序进行不当，则会产生死锁。例如，SI，S2，S3是临时性资源，进程P1产生消息S1，又要求从P3接收消息S3；进程P3产生消息S3，又要求从进程P2处接收消息S2；进程P2产生消息S2，又要求从P1处接收产生的消息S1。

**(3)进程推进顺序不当引起死锁**

由于进程在运行中具有异步性特征。

**4. 处理死锁的方法**

只要上述一个条件不成立，就不会产生死锁，所以解决死锁的基本方法有：预防死锁、避免死锁、检测死锁、解除死锁。

**预防死锁**

首先，预防死锁应该理解它发生的原因，尤其是产生死锁的四个必要条件，就可以最大可能地避免、预防和解除死锁。

**方法一：**只要打破四个必要条件之一

①打破互斥条件（破坏互斥条件）：改造独占性资源， 允许系统资源都能共享使用，则系统不会进入死锁状态。

**缺点：**一般来说互斥条件是无法破坏的，所有的时候还会保护这种情况，例如在打印机打印的时候。(所以在预防死锁时主要从其他三个方面入手)

②可剥夺资源：（破坏不可剥夺条件）当一进程占有一独占性资源后又申请一独占性资源而无法满足，则退出原占有的资源，待以后需要时再重新申请。这意味着，一个进程已占有的资源会被暂时释放，也即是被剥夺了，或从而破坏了不可剥夺条件。

**缺点：**1.反复申请增加系统开销。2.释放已获得的资源可能造成前一阶段工作的失效，这种方法常用于状态易于保存和恢复的资源，如CPU的寄存器及内存资源，一般不能用于打印机之类的资源。

③资源一次性分配：（破坏请求和保持条件）：采用资源预先静态分配策略，即进程运行前申请全部资源，满足则运行，不然就等待，这样就不会占有且申请。

缺点：系统资源被严重浪费，其中有些资源可能仅在运行初期或运行快结束时才使用，甚至根本不使用。而且还会导致“饥饿”现象，当由于个别资源长期被其他进程占用时，将致使等待该资源的进程迟迟不能开始运行。

④打破循环等待条件：（破坏环路等待条件）实现资源有序分配策略(顺序资源分配法)，首先对所有设备实现分类编号，规定每个进程只能采用按序号递增的形式申请资源，同类资源一次申请完。

**缺点：**编号必须相对稳定，这就限制了新类型设备的增加；尽管在为资源编号时已考虑到大多数作业实际使用这些资源的顺序，但也经常会发生作业使用资源的顺序与系统规定顺序不同的情况，造成资源的浪费；此外，这种按规定次序申请资源的方法，也必然会给用户的编程带来麻烦。

所以，当你是一个程序设计者，或者架构师你需要考虑在系统设计、进程调度等方面注意如何不让这四个必要条件成立，如何确定资源的合理分配算法，避免进程永久占据系统资源。有两种方法可以预防死锁：

**避免死锁**

死锁避免的基本思想：系统对进程发出的每一个资源申请进行动态检查，并根据检查结果决定是否分配资源，如果分配后系统可能发生死锁，则不予分配，否则予以分配，这是一种保证系统不进入死锁状态的动态策略。介绍避免死锁的几种常见方法：

1.避免一个线程同时获取多个锁。

2.避免一个线程在锁内同时占用多个资源，尽量保证每个锁只占用一个资源。

3.尝试使用定时锁，使用locak.tryLock(timeout)来代替使用内部锁机制。加锁时限

4.死锁检测：这是一个更好的避免死锁，它主要是针对那些不可能实现按序加锁并且锁超时也不可行的场景。

**5.有序资源分配法**

这种算法资源按某种规则系统中的所有资源统一编号（例如打印机为1、磁带机为2、磁盘为3等等），申请时必须以上升的次序，破坏了环路条件，避免了死锁的发生。

**6.银行家算法**

银行家算法是避免死锁的一种重要方法，防止死锁的机构只能确保上述四个条件之一不出现，则系统就不会发生死锁。通过这个算法可以用来解决生活中的实际问题，如银行贷款等。(先理解到讲算法的时候会讲)

按照银行家算法的思想，当进程请求资源时，系统将按如下原则分配系统资源：

(1) 当一个进程对资源的最大需求量不超过系统中的资源数时可以接纳该进程。

(2) 进程可以分期请求资源，当请求的总数不能超过最大需求量。

(3) 当系统现有的资源不能满足进程该需要资源数时，对进程的请求可以推迟分配，但总能使进程在有限的时间里得到资源。

(4) 当系统现有的资源能满足进程尚需资源数时，必须测试系统现存的资源能否满足该进程尚需的最大资源数，若能满足则按当前的申请量分配资源，否则也要推迟分配。

* **解决死锁的方法**

一般来说解决死锁的方法分为死锁的预防、避免、检测与解除、鸵鸟策略四种（注意：死锁的检测与解除是一个方法）

①死锁预防：这是一种较简单和直观的事先预防的方法。方法是通过设置某些限制条件，去破坏产生死锁的四个必要条件中的一个或者几个，来预防发生死锁。预防死锁是一种较易实现的方法，已被广泛使用。但是由于所施加的限制条件往往太严格，可能会导致系统资源利用率和系统吞吐量降低。

②死锁避免：它不限制进程有关申请资源的命令，而是对进程所发出的每一个申请资源命令加以动态地检查，并根据检查结果决定是否进行资源分配。如果分配后系统可能发生死锁，则不予分配，否则予以分配。这种方法的关键是确定资源分配的安全性，也是一种动态策略。

③死锁检测和解除：先检测：这种方法并不须事先采取任何限制性措施，也不必检查系统是否已经进入不安全区，此方法允许系统在运行过程中发生死锁。检测方法包括定时检测、效率低时检测、进程等待时检测等及时地检测出死锁的发生，并精确地确定与死锁有关的进程和资源。

再解除死锁：常用的实施方法是撤销或挂起一些进程，以便回收一些资源，再将这些资源分配给已处于阻塞状态的进程，使之转为就绪状态，以继续运行。死锁的检测和解除措施，有可能使系统获得较好的资源利用率和吞吐量，但在实现上难度也最大。

④鸵鸟策略：有时当系统发生死锁时不会对用户造成多大影响，或系统很少发生死锁的场合采用允许死锁发生的鸵鸟算法，这样一来可能开销比不允许发生死锁及检测和解除死锁的小。这实在不算是一个算法，但却是目前实际系统采用最多的一种策略。

**五、内存管理**

**5.1 内存管理的方式**

（1）块式管理：把主存分为一大块一大块的，当所需的程序片段不在主存时就分配一块主存空间，把程序片段load入主存，就算所需的程序片段只有几个字节也只能把这一块分配给它。这样会造成很大的浪费，平均浪费了50%的内存空间，但是易于管理。

（2）页式管理：把主存分为一页一页的，每一页的空间要比一块一块的空间小很多，这种方法的空间利用率要比块式管理高很多

（3）段式管理：把主存分为一段一段的，每一段的空间又要比一页一页的空间小很多，这种方法在空间利用率上又比页式管理高得多，但是也有另外一个缺点。一个程序片段可能会被分为几十段，这样很多时间就会被浪费在计算每一段的物理地址上。

（4）段页式管理：结合了段式管理和页式管理的优点。把主存先分成若干段，每个段又分成若干页。段页式管理每取一次数据，要访问3次内存。

**5.2 分页和分段的区别**

页是信息的物理单位，分页是为了实现离散分配方式，以消减内存的外零头，提高内存的利用率；或者说，分页仅仅是由于系统管理的需要，而不是用户的需要。

段是信息的逻辑单位，它含有一组其意义相对完整的信息。分段的目的是为了能更好地满足用户的需要。页的大小固定且由系统确定，把逻辑地址划分为页号和页内地址两部分，是由机器硬件实现的，因而一个系统只能有一种大小的页面。段的长度却不固定，决定于用户所编写的程序，通常由编译程序在对源程序进程编辑时，根据信息的性质来划分。

分页的作业地址空间是一维的，即单一的线性空间，程序员只需利用一个记忆符，即可表示一地址。分段的作业地址空间是二维的，程序员在标识一个地址时，既需给出段名，又需给出段内地址。

**5.3 虚拟内存**

虚拟内存简称虚存，是计算机系统内存管理的一种技术。它是相对于物理内存而言的，可以理解为“假的”内存。它使得应用程序认为它拥有连续可用的内存（一个连续完整的地址空间），允许程序员编写并允许比实际系统拥有的内存大得多的程序，这使得许多大型软件项目能够在具有有限内存资源的系统上实现。而实际上，它通常被分割成多个物理内存碎片，还有部分暂时存储在外部磁盘存储器上，在需要时进行数据交换。虚存比实存有以下好处：

（1） 扩大地址空间。无论段式虚存，还是页式虚存，或是段页式虚存，寻址空间都比实存大。

（2）内存保护。每个进程运行在各自的虚拟内存地址空间，互相不能干扰对方。另外，虚存还对特定的内存地址提供写保护，可以防止代码或数据被恶意篡改。

（3）公平分配内存。采用了虚存之后，每个进程都相当于有太阳大小的虚存空间。

（4）当进程需要通信时，可采用虚存共享的方式实现。

不过，使用虚存也是有代价的，主要表现在以下几个方面：

（1）虚存的管理需要建立很多数据结构，这些数据结构要占用额外的内存。

（2）虚拟地址到物理地址的转换，增加了指令的执行时间

（3）页面的换入换出需要磁盘I/O，这是很耗时间的。

（4）如果一页中只有一部分数据，会很浪费内存。